Herramientas para Computación de Altas Prestaciones

Algoritmos a Bloques

Objetivo de los algoritmos a bloques

Los algoritmos a bloques son variaciones de algoritmos tradicionales, en los que un problema grande se descompone en trozos más pequeños que “caben” en la memoria cache. Se usan para problemas “Grandes”.

Seleccionando cuidadosamente el tamaño de los bloques y el algoritmo, se puede minimizar el tráfico de memoria necesario para resolver el problema.

Todas las implementaciones en librerías de “calidad” de algebra lineal numérica (solución de Sistemas de Ecuaciones Lineales, cálculo de valores y vectores propios, solución de problemas de mínimos cuadrados, etc.) están hechas con Algoritmos a bloques.

Estructura general de los algoritmos a bloques

En los algoritmos típicos del algebra lineal numérica, las versiones a bloques tienen una estructura común. Supondremos que tenemos un solo nivel de cache. Las implementaciones “rápidas” contemplan tantos niveles de “bloques” como niveles de cache haya disponibles.

Sea P el problema que queremos resolver; (Descomposición LU, Desc, Cholesky, resolución de sistemas triangulares, Descomp QR, etc.)

Estructura general de los algoritmos a bloques

Para obtener la versión orientada a bloques del algoritmo P, necesitaremos:

-Una versión del algoritmo P para matrices “que quepan en la cache”; Dicho de otro modo, una versión Pv para matrices “pequeñas”.

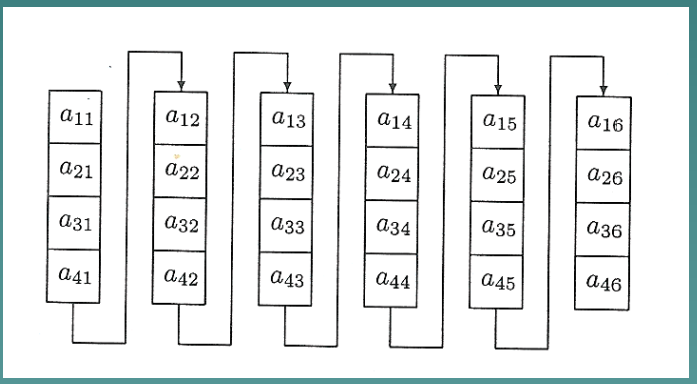
-Una subrutina auxiliar de producto de matrices a bloques

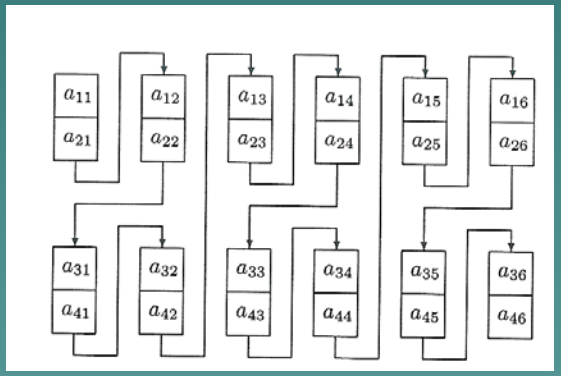
-En algún caso, también subrutinas para resolver sistemas triangulares a bloques

Habitualmente se puede reestructurar el algoritmo P de forma que solo contenga llamadas a la subrutina Pv y a las subrutinas auxiliares

Estructura general de los algoritmos a bloques

En muchos casos, antes de aplicar el algoritmo se copia la matriz a formato “a bloques”.

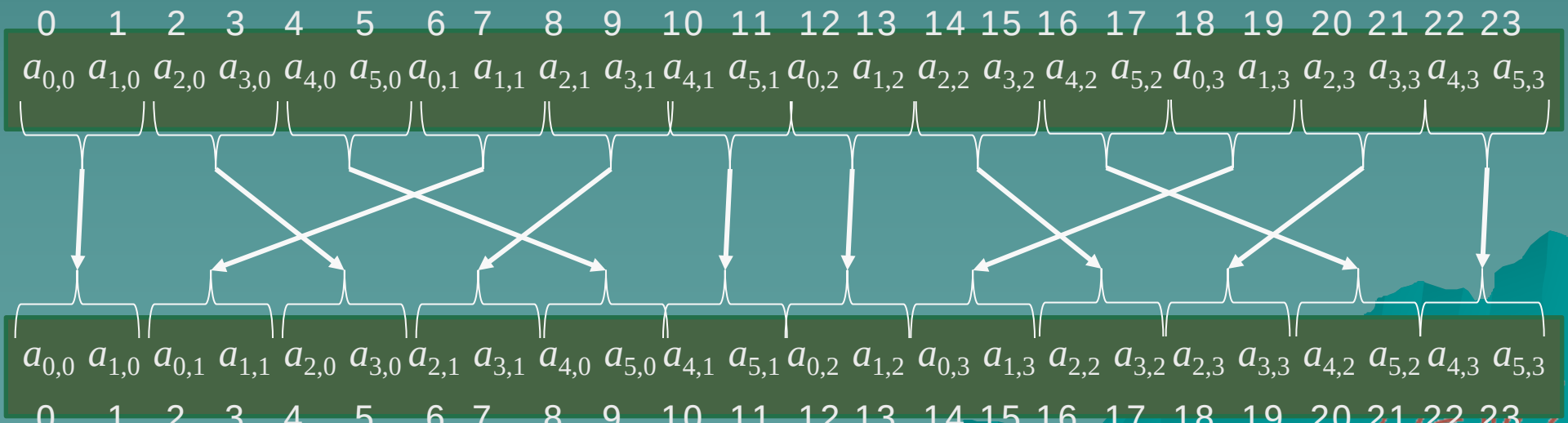




Esto se hace dependiendo del problema a resolver y de la dimensión de las matrices (según si el tiempo gastado en la copia se va a amortizar o no)

Estructura en memoria

$$A = \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \\ a_{4,0} & a_{4,1} & a_{4,2} & a_{4,3} \\ a_{5,0} & a_{5,1} & a_{5,2} & a_{5,3} \end{pmatrix}$$



Notación-1

Sea $A \in \Re^{n*m}$; es posible partir A de la siguiente forma:

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,r} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,r} \\ \vdots & \vdots & \ddots & \vdots \\ A_{q,1} & A_{q,2} & \cdots & A_{q,r} \end{pmatrix} \begin{matrix} m_1 \\ m_2 \\ \\ m_q \end{matrix}$$

$$\begin{matrix} n_1 & n_2 & & n_r \end{matrix}$$

Donde cada bloque $A_{i,j}$ tiene dimensión $m_i \times n_j$;

$m_1 + m_2 + \ldots + m_q = m$ $\qquad$ $n_1 + n_2 + \ldots + n_q = n$

Notación-2

Si B $\in \Re^{n*m}$; :

$$B = \begin{pmatrix} B_{1,1} & B_{1,2} & \cdots & B_{1,r} \\ B_{2,1} & B_{2,2} & \cdots & B_{2,r} \\ \vdots & \vdots & \ddots & \vdots \\ B_{q,1} & B_{q,2} & \cdots & B_{q,r} \end{pmatrix} \begin{matrix} m_1 \\ m_2 \\ \\ m_q \end{matrix}$$

$$\begin{matrix} n_1 & n_2 & & n_r \end{matrix}$$

La partición de B está **conforme** con la de la matriz A (diapositiva anterior)

Notación-3

La suma de C=A+B también será una matriz a bloques con idéntica partición:

$$C = A + B = \begin{pmatrix} C_{1,1} & C_{1,2} & \cdots & C_{1,r} \\ C_{2,1} & C_{2,2} & \cdots & C_{2,r} \\ \vdots & \vdots & \ddots & \vdots \\ C_{q,1} & C_{q,2} & \cdots & C_{q,r} \end{pmatrix} = \begin{pmatrix} A_{1,1}+B_{1,1} & A_{1,2}+B_{1,2} & \cdots & A_{1,r}+B_{1,r} \\ A_{2,1}+B_{2,1} & A_{2,2}+B_{2,2} & \cdots & A_{2,r}+B_{2,r} \\ \vdots & \vdots & \ddots & \vdots \\ A_{q,1}+B_{q,1} & A_{q,2}+B_{q,2} & \cdots & A_{q,r}+B_{q,r} \end{pmatrix}$$

Producto de Matrices-1

Producto de matriz columna a bloques por una matriz fila a bloques: $A \in \Re^{m*p}$, $B \in \Re^{p*n}$

$$A = \begin{pmatrix} A_1 \\ A_2 \\ \vdots \\ A_q \end{pmatrix}; \quad B = \begin{bmatrix} B_1 & B_2 & \cdots & B_r \end{bmatrix}$$

$$A \cdot B = C = \begin{bmatrix} C_{1,1} & C_{1,2} & \cdots & C_{1,r} \\ C_{2,1} & C_{2,2} & \cdots & C_{2,r} \\ \vdots & \vdots & \ddots & \vdots \\ C_{q,1} & C_{q,2} & \cdots & C_{q,r} \end{bmatrix};$$

Donde $C_{i,j} = A_i \cdot B_j$

Producto de Matrices-2

Producto de matriz fila a bloques por una matriz columna a bloques: $A \in \Re^{m*p}$, $B \in \Re^{p*n}$

$$A = \begin{bmatrix} A_1 & A_2 & \cdots & A_s \end{bmatrix}; \quad B = \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_s \end{pmatrix}$$

$$A \cdot B = C = \sum_{j=1}^{s} A_j \cdot B_j;$$

Producto de Matrices-3

Caso general: $A \in \Re^{m*p}$, $B \in \Re^{p*n}$

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,s} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,s} \\ \vdots & \vdots & \ddots & \vdots \\ A_{q,1} & A_{q,2} & \cdots & A_{q,s} \end{pmatrix} \qquad B = \begin{pmatrix} B_{1,1} & B_{1,2} & \cdots & B_{1,r} \\ B_{2,1} & B_{2,2} & \cdots & B_{2,r} \\ \vdots & \vdots & \ddots & \vdots \\ B_{s,1} & B_{s,2} & \cdots & B_{s,r} \end{pmatrix} \begin{matrix} p_1 \\ p_2 \\ \vdots \\ p_s \end{matrix}$$

$$\begin{matrix} p_1 & p_2 & \cdots & p_s \end{matrix}$$

$$A \cdot B = C = \begin{bmatrix} C_{1,1} & C_{1,2} & \cdots & C_{1,r} \\ C_{2,1} & C_{2,2} & \cdots & C_{2,r} \\ \vdots & \vdots & \ddots & \vdots \\ C_{q,1} & C_{q,2} & \cdots & C_{q,r} \end{bmatrix};$$

Dimensiones

Donde $$C_{\alpha,\beta} = \sum_{j=1}^{s} A_{\alpha,j} \cdot B_{j,\beta}$$

Producto matrices a bloques; versión estándar

Supongamos: $A,B,C \in \Re^{n*n}$,$n=N \cdot tb$ ➔ A,B,C son matrices a bloques, con N*N bloques y cada bloque de dimensión tb*tb

$$C_{i,j} = \sum_{k=1}^{s} A_{i,k} \cdot B_{k,j} + C_{i,j}$$

Dado el bloque $C_{i,j}$, ¿Cuáles son sus límites, considerándolo en la "matriz Grande" C?

"Leading dimension"; Cuando se manejan bloques dentro de una matriz mas grande, hay dos dimensiones involucradas, la del bloque y la de la matriz que lo contiene.

Producto matrices a bloques; versión "estándar"

$$C_{i,j} = \sum_{k=1}^{s} A_{i,k} \cdot B_{k,j} + C_{i,j}$$

```
For i=1:N       { N es el número de bloques}
  ind_i=(i-1)*tb+1:i*tb
  For j=1:N
     ind_j=(j-1)*tb+1:j*tb
        For k=1:N
           ind_k=(k-1)*tb+1:k*tb
           C(ind_i,ind_j)=A(ind_i,ind_k)*B(ind_k,ind_j)+C(ind_i,ind_j)
        End
  End
End
```

Computación de Altas Prestaciones

Experimentación con algoritmos a bloques

Algoritmo básico para producto de matrices cuadradas

Vamos a utilizar el archivo test_bloques_col.c (en poliformat) para probar tres versiones del producto de matrices: normal, a bloques “sin copia”, y a bloques “con copia”.

Versión “normal”

```
void matprod(double *A, double *B, double *C, int n)
{    int i,j,k;
     for (j=0; j<n; j++)
        for(k=0;k<n; k++)
           for(i=0;i<n; i++)
              C[i+j*n]= C[i+j*n]+ A[i+k*n]* B[k+j*n];
}
```

Algoritmo a bloques “sin copia” para producto de matrices cuadradas-1

Necesitamos la “Función externa” que funciona “a nivel de bloques” y la Función interna para multiplicar bloques, referenciando los bloques correctamente en las matrices;

Función externa =>

```
void matprodbloques(double *A, double *B, double *C, int lda, int tb )
{    int i,j,k;
     int nb=lda/tb;
     for (j=0; j<nb; j++)
          for(k=0;k<nb; k++)
            for(i=0;i<nb; i++)
              matblockpeq_nocopia(A,B,C,i,j,k,tb,lda);
}
```

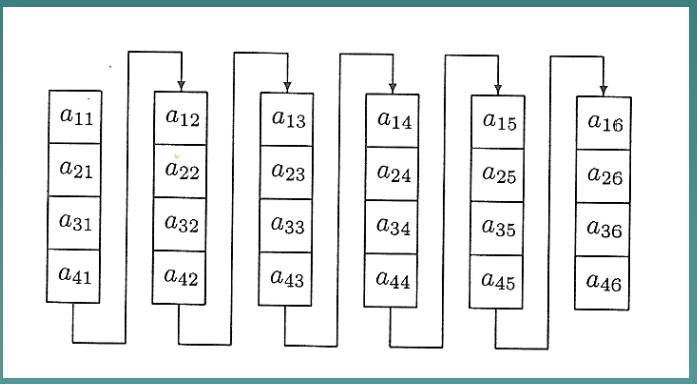
Algoritmo a bloques “sin copia” para producto de matrices cuadradas-1

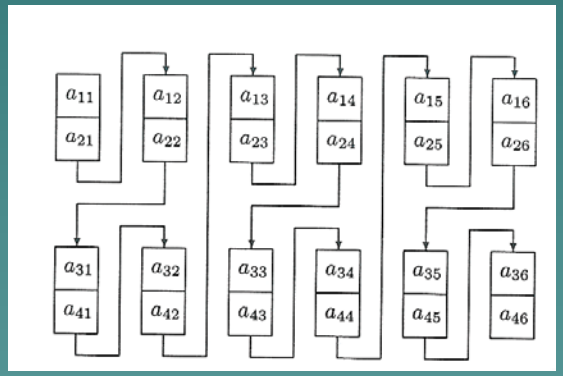
Función interna =>

```
void matblockpeq_nocopia(double *A, double *B, double *C, int
    ib, int jb, int kb, int tb, int lda )
{
    int i,j,k;
    int inii=ib*tb;
    int inij=jb*tb;
    int inik=kb*tb;
    for (j=inij; j<inij+tb; j++)
        for(k=inik;k<inik+tb; k++)
            for(i=inii;i<inii+tb; i++)
         C[i+j*lda]= C[i+j*lda]+ A[i+k*lda]* B[k+j*lda];
}
```

Algoritmo a bloques “con copia”

Copiamos las matrices a formato de bloques





Necesitamos subrutinas para copiar matrices a formato de bloques: To_Blocked, From_Blocked

Algoritmo a bloques “con copia”

Subrutina interna para el caso “con copia”:

```
void Mult_add(double *A, double *B, double *C,int i_bar, int
  j_bar, int k_bar, int n_bar, int tb)
{
  int b_sqr=tb*tb;
  double *c_p = C+(i_bar + j_bar*n_bar)*b_sqr;;
  double *a_p = A + (i_bar + k_bar*n_bar)*b_sqr;
  double *b_p = B + (k_bar+ j_bar*n_bar )*b_sqr;
  int i, j, k;
  for (j = 0; j < tb; j++)
     for (k = 0; k < tb; k++)
        for (i = 0; i < tb; i++)
           c_p[i+j*tb] += a_p[i+k*tb]*(b_p[k+j*tb]);
}
```

Experimentos (1)

La subrutina test_bloques_col.c permite experimentar con los tres algoritmos; el tamaño de bloque (tb) debe ser divisor exacto de la dimensión de las matrices (n).

Experimento 1, comprobar que el producto de matrices se hace correctamente (seleccionar un tamaño pequeño y descomentar las llamadas a la subrutina para escribir la matriz C)

Para los siguientes experimentos, comentaremos de nuevo la subrutina de escritura y seleccionaremos un tamaño grande (1000) y un tamaño de bloque razonable (50).

Experimento 2: compilar con icc y –O3, comparar con gcc y –O3, buscar mejor tamaño de bloques para icc (20, 25, 50, 100, 200)

Experimentos (2)

Experimento 3, Probar los flags –vec-report2, -guide –parallel, -parallel, autooptimización (-prof-gen, -prof-use) combinados con –O3 (solo para icc)

Recuerda que con –parallel el compilador de intel usa su propia subrutina de producto de matrices, se podía comprobar usando gprof.

Valgrind

Valgrind se puede describir como un simulador de CPU en software.

-Su uso principal es el poder detectar errores de memoria típicos de C /C++ (acceso a posiciones de memoria ilegales, memory leaks, ...)

-Se han desarrollado otras herramientas que funcionan bajo valgrind: Cachegrind (cache profiler), HellGrind (Thread debugger), y otros.

www.valgrind.org

Experimentos (3): Estudio de fallos de cache usando valgrind.

Seleccionar un tamaño de matriz n no demasiado grande (no mas de 1000)

Compilar con icc, con –g y con –O3.

Ejecutar el programa “bajo valgrind”:

```
>> valgrind –- tool=cachegrind  nombre_ejecutable
```

Para poder estudiar los accesos a cache de cada uno de los tres algoritmos por separado, comentar las partes apropiadas del programa, recompilar, para ver los fallos de cache.